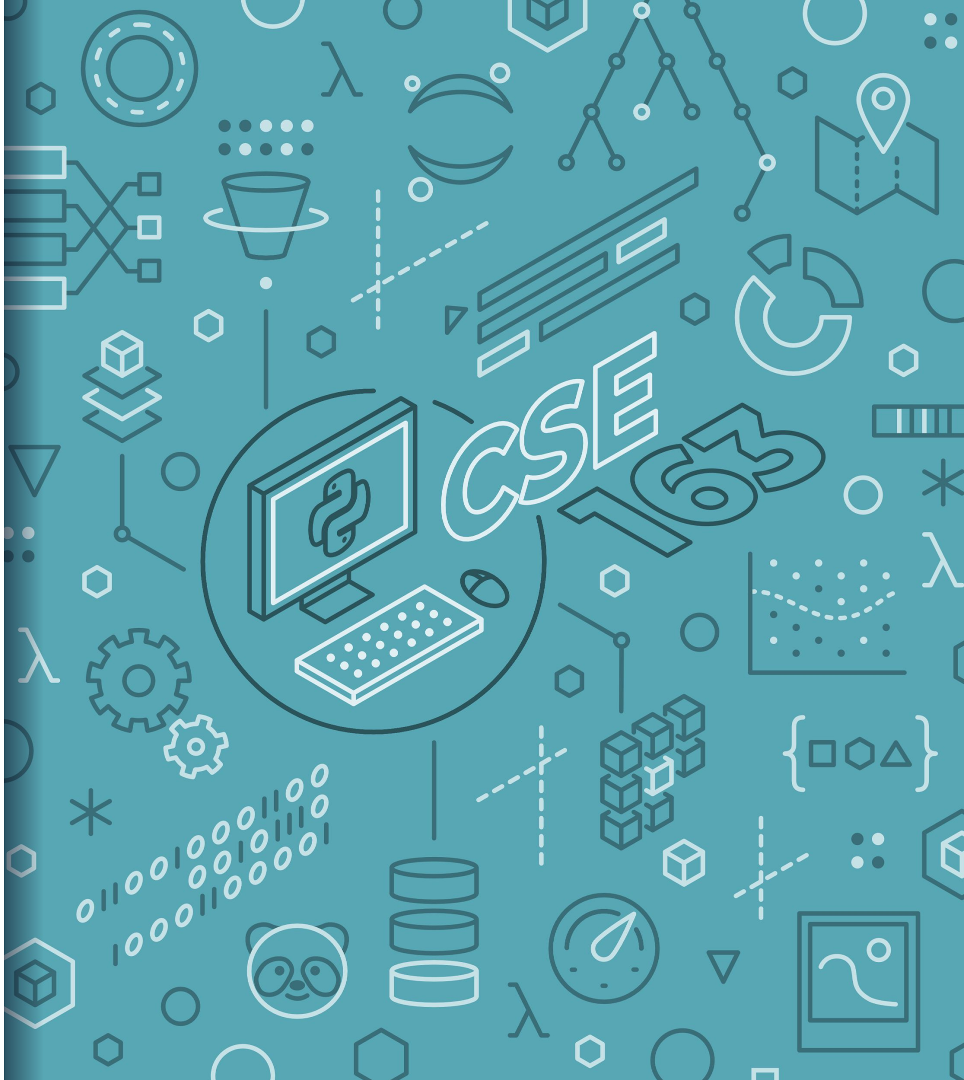
CSE 163

Performance + Profiling

Hunter Schafer



Notecards

- At the end we will be doing an activity to help people to find partners for the project
- If you are still looking for a partner, on the notecard write
  - Your name
  - An area you are most interested in looking into for the project
    - E.g. biology, astronomy, social networks, etc
- We will collect those and talk about an activity at the end

If you already have a partner or don't want one, congrats, you have won a brand new notecard

- Don't turn it in

Method 2: Count steps

Assumption: A single simple line of code takes the same amount of time to run in Python.

Examples:

```
num = 5 + 10
print("hello")
num > 5 and num % 2 == 0
```

Rules:

- Loop runtime is number of times it is run (N) times the number of statements
- Method runtime is sum of all statements found inside of it.

Growth rate

We often think of algorithms in terms of growth rate in terms of the input or input data size (**n**)

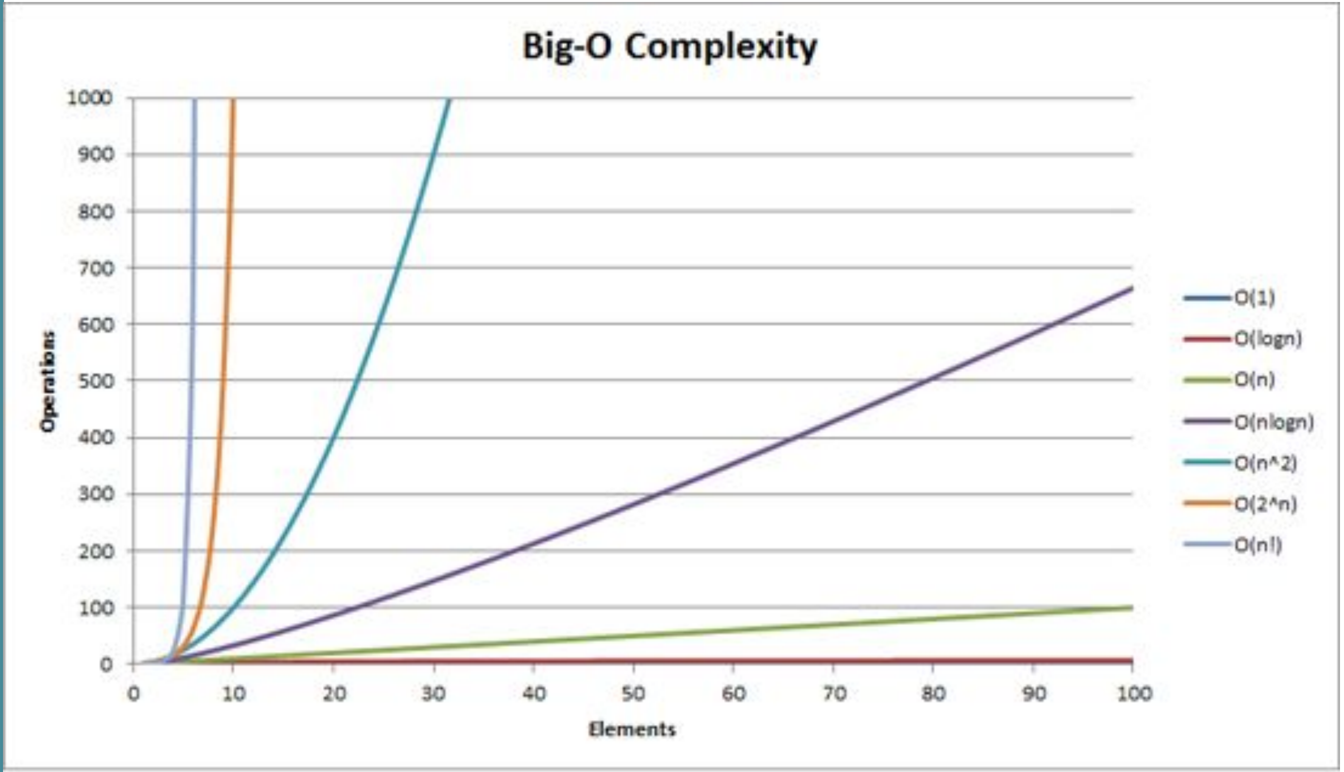
Think of the runtime of $5n^3 + 4n^2 + 3n + 7$.

All terms seem significant when we look at smaller inputs.

However, what happens when n becomes extremely large? At that point the term with the largest power of n will dominate (n^3).

We say it runs on the order of n^3 or $O(n^3)$ (**Big Oh of n cubed**).

Complexity classes



http://recursive-design.com/blog/2010/12/07/comp-sci-101-big-o-notation/ - post about a Google interview

Poll Everywhere

Think

1 min - Review

pollev.com/cse163

What are the Big-O run-times of each of these methods?

```python
def max_diff1(nums):
  max_diff = 0
  for n1 in nums:
    for n2 in nums:
      diff = abs(n1 - n2)
      if diff > max_diff:
        max_diff = diff
  return max_diff
```

```python
def max_diff2(nums):
  min_num = nums[0]
  max_num = nums[0]
  for num in nums:
    if num < min_num:
      min_num = num
    elif num > max_num:
      max_num = num
  return max_num - min_num
```

```python
def max_diff3(nums):
  return max(nums) - min(nums)
```

Poll Everywhere

Pair

1.5 min - Review

pollev.com/cse163

```python
def max_diff1(nums):
  max_diff = 0
  for n1 in nums:
    for n2 in nums:
      diff = abs(n1 - n2)
      if diff > max_diff:
        max_diff = diff
  return max_diff
```

```python
def max_diff2(nums):
  min_num = nums[0]
  max_num = nums[0]
  for num in nums:
    if num < min_num:
      min_num = num
    elif num > max_num:
      max_num = num
  return max_num - min_num
```

```python
def max_diff3(nums):
  return max(nums) - min(nums)
```

Big-O

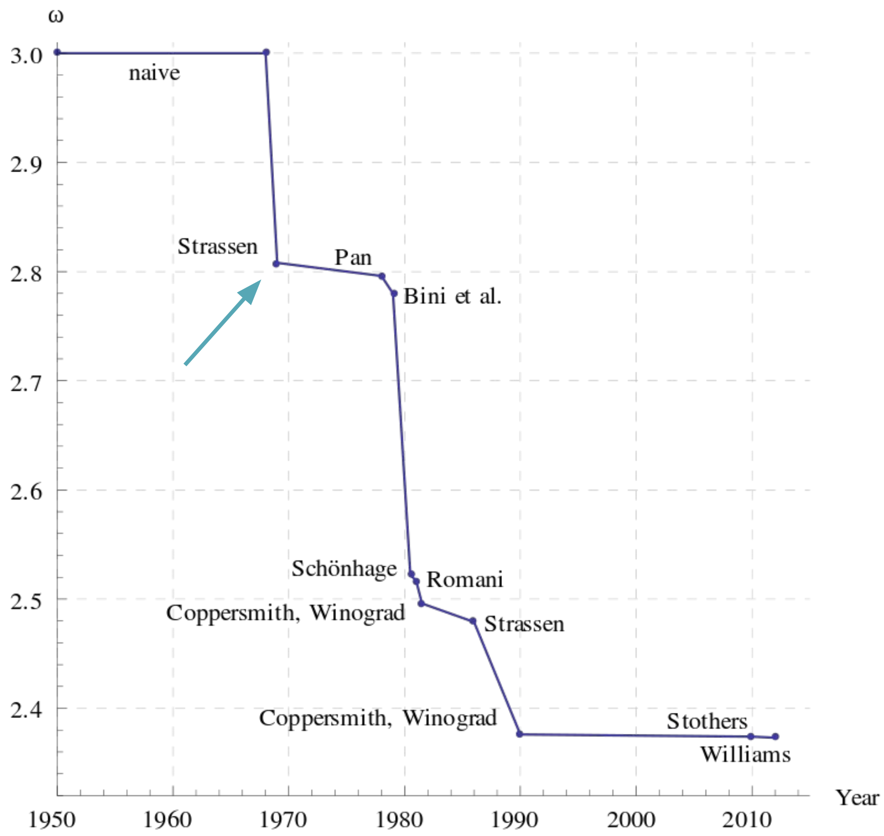
Pros

- Simple to describe how an algorithm will **scale**
- Independent of programming language / computer
- Easy to think of (with practice 😊)

Cons

- Can be a bit overly simplistic (a very crude approximation)
  - Misses many important issues of performance that matter in real life
- Constants can sometimes really matter

Matrix Multiplication



Big-O

- In theory, Big-O is a great descriptor, but in practice we generally also include timing information to help us reason about our program efficiency
- It makes no sense to talk about one or the other, a good computer scientist uses both tools appropriately
    - Big-O
        - Helps us communicate how our algorithm scales
        - Great tool to easily eliminate algorithms that won’t scale
    - Timing
        - Verify our implementation is fast
        - **Help us find bottlenecks in our algorithm**

Poll Everywhere

Think

1 min

pollev.com/cse163

Rank the algorithms in term of their speed on my computer

“1=2 < 3” means 1 and 2 are about the same, but faster than 3

```python
def max_diff1(nums):
  max_diff = 0
  for n1 in nums:
    for n2 in nums:
      diff = abs(n1 - n2)
      if diff > max_diff:
        max_diff = diff
  return max_diff
```

```python
def max_diff2(nums):
  min_num = nums[0]
  max_num = nums[0]
  for num in nums:
    if num < min_num:
      min_num = num
    elif num > max_num:
      max_num = num
  return max_num - min_num
```

```python
def max_diff3(nums):
  return max(nums) - min(nums)
```

Poll Everywhere

Pair

1 min

pollev.com/cse163

Rank the algorithms in term of their speed on my computer

“1=2 < 3” means 1 and 2 are about the same, but faster than 3

```python
def max_diff1(nums):
  max_diff = 0
  for n1 in nums:
    for n2 in nums:
      diff = abs(n1 - n2)
      if diff > max_diff:
        max_diff = diff
  return max_diff
```

```python
def max_diff2(nums):
  min_num = nums[0]
  max_num = nums[0]
  for num in nums:
    if num < min_num:
      min_num = num
    elif num > max_num:
      max_num = num
  return max_num - min_num
```

```python
def max_diff3(nums):
  return max(nums) - min(nums)
```

Timing

- Generally two ways to time programs
  - Time the whole program
  - Use a profiler to help you see the time spent on each line
- If you are timing the whole program, you need to run it multiple times to get a good idea of the total run-time
- If you use a profiler, you usually don't care about the raw times, but rather relative times

- For profiling, I usually use the line_profiler package (kernprof)
- In your terminal

```
conda install line_profiler
kernprof -v -l file.py
```

Why is Python Slow?

Interpreter

- We have mentioned before that when we are using Python, we are using an interpreter rather than a compiler
- This means it has to figure how to translate your code while it is running
- This can be overly slow in “hot” loops

Dynamically types

- We don’t have to write down the types ahead of time, which means Python spends a fair bit of time figuring out which type each object is

Example: Arithmetic

Consider this C program

```c
/* C code */
int a = 1;
int b = 2;
int c = a + b;
```

This roughly maps to the machine instructions

```
Assign <int> 1 to a
Assign <int> 2 to b
call add<int, int>(a, b)
Assign the result to c
```

Example: Arithmetic

Consider this Python program

```python
# A Python program
a = 1
b = 2
c = a + b
```

This roughly maps to the machine instructions

Example: Arithmetic

```
1. Assign 1 to a
  1a. Set a->PyObject_HEAD->typecode to integer
  1b. Set a->val = 1
2. Assign 2 to b
  2a. Set b->PyObject_HEAD->typecode to integer
  2b. Set b->val = 2
3. call add(a, b)
  3a. find typecode in a->PyObject_HEAD
  3b. a is an integer; value is a->val
  3c. find typecode in b->PyObject_HEAD
  3d. b is an integer; value is b->val
  3e. call add<int, int>(a->val, b->val)
  3f. result of this is result, and is an integer.
4. Create a Python object c
  4a. set c->PyObject_HEAD->typecode to integer
  4b. set c->val to result
```

Slow Python

- If Python is so slow, why does anyone use it?
    - It is WAY easier to program with than a language like C
        - Developer time costs $$$, computers are cheap
    - Python provides incredible support for plugging into pre-compiled libraries
        - Examples: pandas, sklearn, numpy, scipy
- Pretty easy to get around this
    - Use a profiler to figure out where your bottlenecks are
    - If your program is slow, look to any “hot” loops that might be replaced with library calls.